Filtering Algorithms for the NValue Constraint

Christian Bessiere[1], Emmanuel Hebrard[2], Brahim Hnich[3], Zeynep Kiziltan[4], and Toby Walsh[2]

[1]LIRMM-CNRS
bessiere@lirmm.fr
[2]NICTA and UNSW
{ehebrard, tw}@cse.unsw.edu.au
[3]Izmir University of Economics
brahim.hnich@ieu.edu.tr
[4]University of Bologna
zkiziltan@deis.unibo.it

Abstract. The NValue constraint counts the number of different values assigned to a vector of variables. Propagating generalized arc consistency on this constraint is NP-hard. We show that computing even the lower bound on the number of values is NP-hard. We therefore study different approximation heuristics for this problem. We introduce three new methods for computing a lower bound on the number of values. The first two are based on the *maximum independent set problem* and are incomparable to a previous approach based on intervals. The last method is a linear relaxation of the problem. This gives a tighter lower bound than all other methods, but at a greater asymptotic cost.

1 Introduction

The NValue constraint counts the number of distinct values used by a vector of variables. It is a generalization of the widely used AllDifferent constraint [5, 16]. It was introduced in [6] to model a musical play-list configuration problem so that play-lists were either homogeneous (used few values) or diverse (used many). There are many other situations where the number of values used are limited. For example, if values represent resources, we may have a limit on the number of values used at the same time. A NValue constraint can thus aid both modelling and solving many real world problems.

Enforcing generalized arc consistency (GAC) on the NValue constraint is NP-hard [3]. One way to deal with this intractability is to decompose the constraint or to approximate the pruning. The NValue constraint can be decomposed into two other global constraints: the AtMostNValue and the AtleastNValue constraints. Unfortunately, while enforcing GAC on the AtLeastNValue constraint is polynomial, enforcing GAC on the AtMostNValue constraint is also NP-hard. We will therefore focus on various approximation methods for propagating the AtMostNValue constraint.

We introduce three new approximations. Two are based on graph theory while the third exploits a linear relaxation encoding. We compare the level of filtering achieved with a previous approximation method due to Beldiceanu based on intervals that runs in $O(n\log(n))$ [1]. We show that the two new algorithms based on graph theory are incomparable with Beldiceanu's, though one is strictly tighter than the other. Both algorithms, however, have an $O(n^2)$ time complexity. We also show that the linear relaxation method dominates all other approaches in terms of the filtering, but with a higher computational cost. Finally, we demonstrate how all of these methods can be used in a filtering algorithm for the NValue constraint and provide empirical results to show the value of the methods proposed.

2 Formal Background

2.1 Constraint satisfaction problems

A constraint satisfaction problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints that specify allowed combinations of values for subsets of variables. We use upper case for variables, X_i, or vectors of variables, $\bar{X}$, and lower case for values, v, or assignments, $\bar{v}$. The domain of a variable X_i, $D(X_i)$ is a set of values. A full or partial assignment $\bar{v} = \langle v_1, \ldots, v_m\rangle$ of $\bar{X} = \langle X_1, \ldots, X_m\rangle$ is a vector of values such that $v_i \in D(X_i)$. A solution to a CSP is a full assignment of values to the variables satisfying the constraints. The minimum (resp. maximum) value in the domain of a variable X_i is $min(X_i)$ (resp. $max(X_i)$). The cardinality of an assignment $\bar{v}$ is $card(\bar{v})$, the number of distinct values used. For instance if $\bar{v} = \langle a, b, a, b, c\rangle$, $card(\bar{v}) = 3$. The maximum (resp. minimum) cardinality of a vector of variables $\bar{X}$, $card{\uparrow}(\bar{X})$ (resp. $card{\downarrow}(\bar{X})$) is the largest (resp. smallest) cardinality among all possible assignments.

Constraint solvers typically explore partial assignments enforcing a local consistency property using either specialized or general purpose propagation algorithms. Given a constraint C on the variables $\bar{X}$, a *support* for $X_i = v_j$ on C is a partial assignment $\bar{v}$ of $\bar{X}$ containing $X_i = v_j$ that satisfies C. A value $v_j \in D(X_i)$ without support on a constraint is *arc inconsistent*. A variable X_i is *generalized arc consistent* (GAC) on C iff every value in $D(X_i)$ has support on C. A constraint C is GAC iff each constrained variable is GAC on C. A *bound support* on C is a support where the interval $[min(X_i), max(X_i)]$ is substituted for the domain of each constrained variable X_i. A variable X_i is *bound consistent* (BC) on C if $min(X_i)$ and $max(X_i)$ have bound support on C. A constraint is BC iff all constrained variables are BC on C.

In line with [4], we say that a local consistency property Φ on C is as strong as Ψ (written $\Phi \succeq \Psi$) iff, given any domains, if Φ holds then Ψ holds; Φ is stronger than Ψ (written $\Phi \succ \Psi$) iff $\Phi \succeq \Psi$ but not $\Psi \succeq \Phi$; Φ is equivalent to Ψ (written $\Phi \equiv \Psi$) iff $\Phi \succeq \Psi$ and $\Psi \succeq \Phi$; and that they are incomparable otherwise (written $\Phi \bowtie \Psi$).

2.2 Graph theoretic concepts

Given a family of sets $\mathcal{F} = \{S_1, \ldots, S_n\}$ and a graph $G = (V, E)$ with the set of vertices $V = \{v_1, \ldots, v_n\}$ and set of edges E, G is the *intersection graph of* $\mathcal{F}$ iff

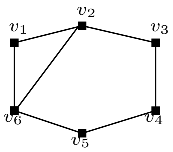
$$\forall i, j \cdot \langle v_i, v_j \rangle \in E \leftrightarrow S_i \cap S_j \neq \emptyset$$

For any graph G, there exists a family of sets $\mathcal{F}$ such that the intersection graph of $\mathcal{F}$ is G. The class of intersection graphs is simply the class of all undirected graphs [12]. The class of graphs obtained by the intersection of *intervals*, instead of sets, is known as *interval graphs*.

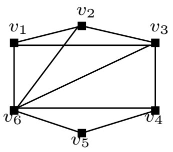
Given a vector of variables $\bar{X} = \langle X_1, \ldots, X_m \rangle$, we use $G_{\bar{X}} = (V, E)$ for the induced intersection graph, i.e the graph where $V = \{v_1, \ldots, v_m\}$ and $\forall i, j \cdot \langle v_i, v_j \rangle \in E \leftrightarrow D(X_i) \cap D(X_j) \neq \emptyset$. Similarly, we use $\bar{I}$ for the same vector of variables, where all domains are seen as intervals instead, i.e., for each i, $D(X_i) = [min(X_i), max(X_i)]$. $G_{\bar{I}}$ is the induced interval graph, defined like $G_{\bar{X}}$, but on the intervals instead. For instance, the domains in Figure (1,a) induce the intersection graph in (1,b) and the interval graph in (1,c).

$X_1 \in \{2, 3\}$
$X_2 \in \{3, 4\}$
$X_3 \in \{1, 4, 5\}$
$X_4 \in \{5, 6\}$
$X_5 \in \{6, 7\}$
$X_6 \in \{2, 3, 7\}$

(a) domains



(b) $G_{\bar{X}}$



(c) $G_{\bar{I}}$

Fig. 1. Domains, intersection graph and interval graph.

Finally, we recall that an *independent set* is a set of vertices with no edge in common. The *independence number* $\alpha(G)$ of a graph G, is the number of vertices in an independent set of maximum cardinality. A *clique* is the dual concept: a set of vertices such that any pair are connected by an edge. A *clique cover* of G is a partition of the vertices into cliques. The cardinality of the *minimum clique cover* is $\theta(G)$. For instance, the interval graph of Figure (1,c) has $\{\{v_1, v_2, v_3\}\{v_4, v_5, v_6\}\}$ as a minimal clique cover, hence $\theta(G_{\bar{I}}) = 2$. Similarly, the intersection graph of Figure (1,b) has $\{v_1, v_3, v_5\}$ as a maximal independent set, hence $\alpha(G_{\bar{X}}) = 3$.

3 The NValue constraint

In this section we define the NValue constraint and we show that it can be decomposed into two simpler constraints. Whereas one of these constraints is polynomial to propagate using a maximum matching algorithm, the second is NP-hard so we look at approximation methods.

Definition 1. NValue$(N, [X_1, \ldots, X_m])$ *holds iff* $N = |\{X_i| \ 1 \le i \le m\}|$

Enforcing GAC on the NValue constraint is NP-hard in general [3]. We can, however, decompose it into two simpler constraints: the AtLeastNValue and the AtMostNValue constraints.

Definition 2. AtLeastNValue$(N, [X_1, \ldots, X_m])$ *holds iff* $N \le |\{X_i| \ 1 \le i \le m\}|$. AtMostNValue$(N, [X_1, \ldots, X_m])$ *holds iff* $N \ge |\{X_i| \ 1 \le i \le m\}|$.

We can identify precisely when the decomposition of a NValue constraint does not hinder propagation. In order to prove Theorem 1 we first state the two following lemmas.

Lemma 1. *Any value in* $D(N)$ *is GAC for* NValue *if and only if it is lower than or equal to* $card{\uparrow}(\bar{X})$ *and greater than or equal to* $card{\downarrow}(\bar{X})$.

Proof. Let S be any assignment of $\bar{X}$. Consider assigning $\bar{X}$ as in S, one variable at a time. Let $\bar{X}_k$ be $\bar{X}$ at step k, that is, with k ground variables. Hence, since $\bar{X}$ involves m values, $\bar{X}_m$ corresponds to S. At a step k, the value of $card{\downarrow}(\bar{X}_k)$ (resp. $card{\uparrow}(\bar{X}_k)$) increases (resp. decreases) by at most one with respect to step $k-1$. Moreover, when every variable is assigned, $card{\downarrow}(\bar{X}_m) = card{\uparrow}(\bar{X}_m) = card(S)$. Therefore, for any value p between $card{\downarrow}(\bar{X}_0)$ and $card{\uparrow}(\bar{X}_0)$, there exists k such that either $card{\downarrow}(\bar{X}_k) = p$ or $card{\uparrow}(\bar{X}_k) = p$. Consequently p has a support for a sub-domain $\bar{X}_k$ and is thus GAC. □

Second, the variables in $\bar{X}$ are GAC if either $D(N) = [card{\downarrow}(\bar{X}), card{\uparrow}(\bar{X})]$ or there exists at least one value lower than $card{\uparrow}(\bar{X})$ and greater than $card{\downarrow}(\bar{X})$.

Lemma 2. *If either* $D(N) = [card{\downarrow}(\bar{X}), card{\uparrow}(\bar{X})]$ *or* $card{\downarrow}(\bar{X})+1 < card{\uparrow}(\bar{X})$ *and* $[card{\downarrow}(\bar{X})+1, card{\uparrow}(\bar{X})-1] \cap D(N) \neq \emptyset$ *then* $\bar{X}$ *is GAC.*

Proof. We first show the first part of the disjunction. Recall that $card{\downarrow}(\bar{X})$ (resp. $card{\uparrow}(\bar{X})$) is the cardinality of the smallest (resp. largest) possible assignment. Therefore, if the domain of N is equal to the interval $[card{\downarrow}(\bar{X}), card{\uparrow}(\bar{X})]$ it means that all assignments of $\bar{X}$ have a cardinality in $D(N)$.

For the second part, we use again the argument that assigning a single variable can affect the bounds by at most one. In other words, for all $X_i \in \bar{X}$, a value $v \in D(X_i)$ (without loss of generality) belongs to an assignment of cardinality either $card{\downarrow}(\bar{X})$, $card{\downarrow}(\bar{X})+1$, $card{\uparrow}(\bar{X})$ or $card{\uparrow}(\bar{X})-1$. Moreover, let $\bar{X}_{X_i=v}$ be $\bar{X}$ where the domain of X_i is reduced to $\{v\}$. We have $card{\downarrow}(\bar{X}_{X_i=v}) \le card{\downarrow}(\bar{X})+1$ and $card{\uparrow}(\bar{X}_{X_i=v}) \ge card{\uparrow}(\bar{X})-1$. Hence, by assumption $D(N) \cap [card{\downarrow}(\bar{X}_{X_i=v}), card{\uparrow}(\bar{X}_{X_i=v})] \neq \emptyset$, and by applying Lemma 1, we know that there exists a tuple satisfying NValue with $X_i = v$. □

Theorem 1. *If* AtLeastNValue *and* AtMostNValue *are GAC and* ($|D(N)| \neq 2$ *or* $min(N)+1 = max(N)$)*, then* NValue *is GAC.*

Proof. Suppose that the decomposition is GAC. Then we have $card{\downarrow}(\bar{X}) \le min(N)$ and $card{\uparrow}(\bar{X}) \ge max(N)$. Thus, by Lemma 1, N is GAC for NValue.

Furthermore, we know that if $D(N)$ contains a value v such that $card{\downarrow}(\bar{X}) < v < card{\uparrow}(\bar{X})$, then all variables in $\bar{X}$ are GAC (see Lemma 2). Therefore we only need to cover the three cases where $D(N)$ does not contain any value between $card{\downarrow}(\bar{X})$ and $card{\uparrow}(\bar{X})$:

- $D(N) = \{card{\uparrow}(\bar{X})\}$. Let v be an arc inconsistent value in $\bar{X}$. There is no assignment whose cardinality is greater than $card{\uparrow}(\bar{X})$, therefore v is arc inconsistent because it participates only in assignments of cardinality below N. Hence v is arc inconsistent for AtLeastNValue, which contradicts the hypothesis.
- $D(N) = \{card{\downarrow}(\bar{X})\}$. Analogous to the previous case.
- $D(N) = \{card{\downarrow}(\bar{X}), card{\uparrow}(\bar{X})\}$: $|D(N)|$ is not different from 2, so by assumption, $min(N) + 1 = max(N)$, which implies that $card{\downarrow}(\bar{X}) + 1 = card{\uparrow}(\bar{X})$. Then NValue is GAC (see Lemma 2).

$\square$

The only case where AtMostNValue and AtLeastNValue are GAC but NValue may not be is when the domain of N contains only $card{\downarrow}(\bar{X})$ and $card{\uparrow}(\bar{X})$ and there is a gap between these bounds. For instance, consider the domains: $X_1 \in \{1, 2, 3\}, X_2 \in \{1, 2\}, X_3 \in \{1\}, N \in \{1, 3\}$. Whilst enforcing GAC on NValue(X_1, X_2, X_3, N) will prune $X_1 = 2$, these domains are GAC for the decomposition. In section 7, we show that we can make GAC on the decomposition equivalent, by performing extra pruning in this situation.

3.1 The AtLeastNValue constraint

We first have a brief look at the AtLeastNValue constraint. It is known [1] that $card{\uparrow}(\bar{X})$ is the cardinality of the maximal matching of the bipartite graph with a class of vertices representing the variables, another the values, and where an edge links two vertices if and only if it corresponds to a valid assignment. Indeed, this is the basic idea behind Régin's algorithm for enforcing GAC on the AllDifferent constraint [16]. We can easily derive a propagation procedure for AtLeastNValue using the *variable-based* violation cost for the SoftAllDiff constraint as described in [14]. This violation cost counts the number of variables that need to be reassigned to satisfy the constraint and is thus equal to $n - card{\uparrow}(\bar{X})$. The value of $card{\uparrow}(\bar{X})$ is shown in [14] to be equal to the size of the maximal matching in the bipartite graph described above. Moreover, we can prune the values in $\bar{X}$ that do not belong to a maximal matching. This provides us with an algorithm for enforcing GAC on AtLeastNValue. One difference is that we do not always want to prune the values that do not participate in a maximal matching. We shall see how the method described in [14] can be used when pruning the variables in $\bar{X}$ in section 7. We refer the reader to [14] for more details about this algorithm, and we focus on the constraint AtMostNValue for the rest of the paper.

3.2 The AtMostNValue constraint

We adapt the proof of NP-hardness for NValue [3] to show that enforcing GAC on an AtMostNValue constraint alone is intractable.

Theorem 2. *Enforcing GAC on an* AtMostNValue$(N, [X_1, \ldots, X_{m+k}])$ *constraint is NP-hard, and remains so even if N is ground.*

Proof. We use a reduction from 3SAT. Given a formula in k variables and m clauses, we construct the AtMost$N$Value$(N, [X_1, \ldots, X_{m+k}])$ constraint in which $D(X_i) = \{i, \neg i\}$ for all $i \in [1, k]$, and each X_i for $i > k$ represents one of the m clauses. If the jth clause is $x \vee \neg y \vee z$ then $D(X_{k+j}) = \{x, \neg y, z\}$. By construction, the variables $[X_1, \ldots, X_k]$ will consume k distinct values, hence if $N = k$, the constructed AtMost$N$Value constraint has a solution iff the original 3SAT problem has a satisfying assignment. Consider for instance the following 3SAT formula ϕ:

$$\begin{aligned} \phi = c_1 &: a \vee \neg b \vee c \ \wedge \\ c_2 &: \neg a \vee b \vee d \ \wedge \\ c_3 &: \neg b \vee \neg c \vee d \end{aligned}$$

We create an instance of AtMostNValue with 7 variables, 4 standing for atoms and 3 standing for clauses, as follows:

$$\begin{aligned} X_1 \in \{a, \neg a\},\ X_2 \in \{b, \neg b\},\ X_3 \in \{c, \neg c\},\ X_4 \in \{d, \neg d\} \\ X_5 \in \{a, \neg b, c\},\ X_6 \in \{\neg a, b, d\},\ X_7 \in \{\neg b, \neg c, d\} \end{aligned}$$

Any assignment of $\langle X_1, X_2, X_3, X_4 \rangle$ corresponds to an interpretation of ϕ, and uses exactly 4 different values. Therefore, such an interpretation is a model of ϕ if and only if it intersects the domain of every variable standing for a clause, i.e., an assignment using no more than 4 values exists. It follows that testing a value for support is NP-complete, and enforcing GAC is NP-hard. □

Note that this proof is a reduction of 3SAT into the problem of propagating GAC on $\bar{X}$ when N is ground. This means that pruning $\bar{X}$ alone is NP-hard. Indeed, even computing just the lower bound on N, given $\bar{X}$ is not easier.

Theorem 3. *Computing the value of $card{\downarrow}(\bar{X})$ is NP-hard.*

Proof. Computing $card{\downarrow}(\bar{X})$ is equivalent to finding the cardinality of a *minimum hitting set* of $\bar{X}$ seen as a family of sets. A *hitting set* of a family of sets $\mathcal{F}$, is a set that intersects each member of $\mathcal{F}$. Computing the cardinality of the smallest possible hitting set is NP-hard [9]. If we have one variable X_i in $\bar{X}$ for each set $S_i \in \mathcal{F}$, and $D(X_i) = S_i$, then $card{\downarrow}(\bar{X})$ is equal to the cardinality of a minimum hitting set of $\mathcal{F}$. □

4 Existing algorithm for the AtMostNValue constraint

We first recall Beldiceanu's algorithm [1] in figure 2, then we introduce a graph theoretic view of his method. We shall refer to Beldiceanu's algorithm as `OI`, for *ordered intervals*. The first step is to order the domains by increasing lower bound. Then the following procedure (algorithm 1) can be applied, the value returned ($N_{distinct}$) is a lower bound on $card\downarrow(\bar{X})$.

Algorithm 1: `OI`

```
Data   : X̄
Result : N_distinct
N_distinct ← 1;
reinit ← true;
i ← 1;
low ← −∞;
up ← ∞;
while i < m do
    if ¬reinit then i ← i + 1;
    if reinit or (low < min(X_i)) then low ← min(X_i);
    if reinit or (up > max(X_i)) then up ← max(X_i);
    reinit ← (low > up);
    if reinit then N_distinct ← N_distinct + 1;
return N_distinct;
```

Fig. 2. The interval-based algorithm introduced in [1].

The intervals are explored one at a time, and a new group, i.e. a clique of the interval graph, is completed when an interval is found that does not overlap with all previous ones in the group. The time complexity is $O(n \log(n))$ for sorting, and then the algorithm itself is linear, the loop visits each domain at most twice (when this domain is distinct from the previous). Hence, the worst case time complexity is dominated by $O(n \log(n))$. This algorithm is proved correct, that is, it returns a valid lower bound, by noticing that the intervals with smallest maximum value for each group are pairwise disjoint. Consequently, at least as many values as groups, that is, $N_{distinct}$, have to be used. As there was no proof given in [1], we present one here:

Proposition 1. *Let $\{C_1, \ldots, C_k\}$ be a partition of the intervals, output of* `OI`*. If $\bar{I} = \langle I_1, \ldots, I_k \rangle$ is the vector of intervals where I_i is the element of C_i with least maximum value, then all elements of $\bar{I}$ have empty pairwise intersections.*

Proof. `OI` scans all intervals by increasing lower bound, partitioning into groups on the way. When the algorithm ends, we have k groups $C_1, \ldots, C_k$. For any group C_i, consider the interval I_1 with least upper bound. This interval does not intersect any interval in any group C_j such that $j > i$. Suppose it was the case, i.e, there exists $I_2 \in C_j$ which intersects with I_1, since the intervals are ordered by increasing lower bound, I_2 cannot be completely *below* any interval in C_i. It must then be either completely *above* or overlapping. However, since I_l has the least upper bound and intersects I_2, all intervals in C_i must also intersect I_2. It follows that I_2 should belong to C_i hence the contradiction. The set containing the interval with least upper bound of every group is then pairwise disjoint, and is of cardinality k. □

Moreover, it is easy to see that, when the domains are intervals, this bound can be achieved. If, for each group, we assign all the variables of this group to one of the common values, then we obtain an assignment of cardinality $N_{distinct}$. This argument is used in [1] to show that `OI` achieves BC on N.

Now, recall that $G_{\bar{X}}$ is the intersection graph of the variables in $\bar{X}$, whereas $G_{\bar{I}}$ is the interval graph of the same variables. It is easy to see that `OI` computes at once a clique cover and an independent set of $G_{\bar{I}}$. Moreover, since for any graph $\alpha(G) \leq \theta(G)$, if a graph G contains an independent set *and* a clique cover of cardinality n, we must conclude that $n = \alpha(G) = \theta(G)$. Indeed, interval graphs belong to the class of perfect graphs, for which, by definition, the independence number is equal to the size of the minimum clique cover. Therefore, we know that the output of `OI`, i.e., $N_{distinct}$ is equal to $\alpha(G_{\bar{I}})$ and also to $\theta(G_{\bar{I}})$. It can be shown that, in this case, the cardinality of the minimum clique cover on the interval graph is equal to the cardinality of the minimum hitting set on $\bar{I}$ itself. This is due to the fact that a set of intervals that pairwise intersect always share a common interval, any element of this interval hitting all of them. To summarize, in the special case where the domains of all variables in $\bar{X}$ are intervals (denoted $\bar{I}$), the following equality holds: $\alpha(G_{\bar{I}}) = \theta(G_{\bar{I}}) = card{\downarrow}(\bar{I})$.

As a consequence, the value $N_{distinct}$ is exact, hence `OI` achieves bound consistency on N for the constraint AtMostNValue. However, considering domains as intervals may be in some case a very crude approximation. If we consider the intersection graph $G_{\bar{X}}$ instead of the interval graph $G_{\bar{I}}$, the relation becomes: $\alpha(G_{\bar{X}}) \leq \theta(G_{\bar{X}}) \leq card{\downarrow}(\bar{X})$

Any of those three quantities is a valid lower bound, though they are NP-hard to compute. They are, on the other hand, tighter approximations that do not consider domains as intervals. Indeed, since $G_{\bar{X}}$ has less edges than $G_{\bar{I}}$, it follows immediately that: $\alpha(G_{\bar{X}}) \geq \alpha(G_{\bar{I}})$ $(= \theta(G_{\bar{I}}) = card{\downarrow}(\bar{I}))$.

5 Three new approaches

We present two algorithms approximating $\alpha(G_{\bar{X}})$ and a linear relaxation approximating directly the minimum hitting set problem, and hence $card{\downarrow}(\bar{X})$.

5.1 A greedy approach

We have seen that `OI` approximates the lower bound on N by computing the exact value of $\alpha(G_{\bar{I}})$, the independence number of the interval graph induced by $\bar{X}$. Here the idea is to compute the independence number of $G_{\bar{X}}$, $\alpha(G_{\bar{X}})$.

Whilst computing the exact value of $\alpha(G_{\bar{X}})$ is intractable for unrestricted graphs, some efficient approximation schemes exist for that problem. We use here a very simple heuristic algorithm for computing the independence number of a graph, referred to as "the natural greedy algorithm". We denote it `MD`, for *minimum degree*. This procedure, represented in figure 3, consists of removing the vertex v of minimum degree as well as its neighborhood $\Gamma(v)$ in turn. We define the neighborhood of a vertex as the set of vertices sharing an edge with it ($\Gamma(v) = \{w \mid (vw) \in E\}$).

Algorithm 2: `MD`

Data : $G = (V, E)$
Result : $N_{distinct}$
if $G = \emptyset$ **then** return 0;
choose $v \in V$ such that $d(v)$ is minimum;
return 1+MD($G = (V \setminus (\Gamma(v) \cup \{v\}), E)$);

Fig. 3. A greedy algorithm approximating the maximum independent set of a graph.

The number of iterations i is such that $i \leq \alpha(G)$. This algorithm is studied in detail in [10]. If we suppose that the intersection graph is constructed once and maintained during search, then an efficient implementation runs in $O(n+m)$ where n is the number of vertices and m is the number of edges (linear in the size of the graph). However, computing the intersection graph requires $n(n+1)/2$ tests of intersection. Each of those may require at most d equality checks, where d is the size of the domains in $\bar{X}$. Notice that efficient data structures, such as bit vectors, are often used to represent domains and thus allow intersection checks in almost constant time in practice. This suggests an implementation where the graph is never actually computed, but an intersection check is done each time we need to know if an edge links two nodes. The worst case time complexity is then $O(dn^2)$ if intersection is linear in the size of the sets or $O(n^2)$ if it is constant.

5.2 Turán's approximation

One alternative is to use an even simpler approximation to the independence number. Turán proposed a lower bound for $\alpha(G)$ in [17], where n is the number of vertices and m the number of edges (figure 4, relation 1). This inequality was subsequently improved by Favaron *et al.* [7], see relation 2 in figure 4. However, we restrict our study to the original lower bound of Turán (relation 1). Assuming that m is computed once, and revised whenever a domain changes or whenever the constraint is called again, this formula gives a lower bound in constant time. The worst case time complexity is the same as `MD`'s (because of the initialization). However, this heuristic can be much more efficient in practice. We refer to this method as `Turan`.

$$\alpha(G) \geq \left\lceil \frac{n^2}{2m+n} \right\rceil \qquad (1) \qquad\qquad \alpha(G) \geq \left\lceil \frac{2n - \frac{2m}{\lceil \frac{2m}{n} \rceil}}{\lceil \frac{2m}{n} \rceil + 1} \right\rceil \qquad (2)$$

Fig. 4. Graph theoretic lower bounds on the independence number of a graph

5.3 A linear relaxation approach

We have seen that the cardinality of the minimum hitting set problem where the family of sets is formed by the domains of the variables in $\bar{X}$ is equal to the lower bound on N, that is, $card{\downarrow}(\bar{X})$. One difficulty is that approximation algorithms proposed in the literature for minimum hitting set return an approximation by above, and so do not provide a valid lower bound. However, we consider here a linear relaxation that can be solved in polynomial time that gives a lower bound on the minimum hitting set cardinality, and thus, of N. Given a vector of variables $\bar{X} = \langle X_1, \dots X_n \rangle$, let $V = \bigcup_{v \in \bar{X}} D(x)$ be the total set of values. Then let $\{y_v |\ v \in V\}$ be a set of linear variables, and the program, denoted `LP`, is shown in figure 5.

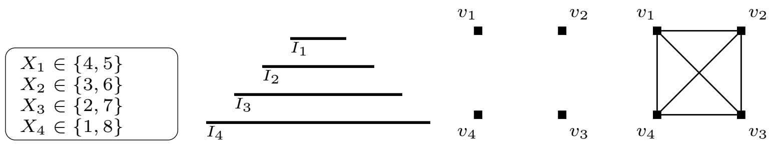
$$\begin{array}{c} \texttt{LP}: \ min \sum_{v \in V} y_v \ \text{ subject to} \\ \sum_{v \in D(X_i)} y_v \geq 1 \ \ \forall X_i \in \bar{X} \\ y_v \geq 0 \ \ \forall v \in V \end{array}$$

Fig. 5. Linear program to approximate the minimum hitting set problem

The best polynomial linear program solvers based on the interior point methods run in $O(v^3 L)$ where v is the number of variables and L is the number of bits in the input. The number of variables in our linear program is nd ($d = |D(X_i)|$) and we have $n = |\bar{X}|$ inequalities of size d. Therefore, the worst case time complexity is $O(n^4 d^4)$. In practice, the simplex method may behave better even though it has an exponential worst case time complexity.

6 Theoretical Analysis

We will compare the level of local consistency computed by these different methods. We refer to the level of local consistency achieved by an algorithm A as A as well. The AtMostNValue constraint is locally consistent iff the lower bound returned by the propagation algorithm does not exceed $max(N)$. $\Phi \succeq \Psi$ means that the lower bound on N returned by the algorithm enforcing Φ is greater



(a) domains (b) intervals (c) $G_{\bar{X}}$ (d) $G_{\bar{I}}$

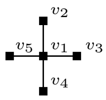
Fig. 6. Example for `OI` $\not\succeq$ `MD` and for `OI` $\not\succeq$ `Turan`.

than or equal to the lower bound returned by the algorithm enforcing Ψ. We will compare the level of local consistency achieved by the following algorithms: `OI`, `MD`, `Turan`, and `LP`.

Note that, since only the lower bound on N is considered in this comparison, `OI` is then equivalent to BC. We do not compare with generalized arc consistency either, as this is NP-hard to enforce and all our algorithms are polynomial and strictly weaker.

Theorem 4. `MD` $\succ$ `Turan`

Proof. For a proof that `MD` is as strong as `Turan` see [10]. Moreover, it is easy to find an example showing that `MD` is strictly stronger. For instance consider the following domains: $X_1 \in \{1,2,3,4,5,6,7,8\},\ X_2 \in \{1,2\},\ X_3 \in \{3,4\},\ X_4 \in \{5,6\}\ ,\ and\ X_5 \in \{7,8\}$. The induced intersection graph is as follows:



When applying `MD`, we obtain an independent set of size 4. However, `Turan` returns: $\lceil \frac{n^2}{2m+n} \rceil = \lceil \frac{25}{13} \rceil = 2$. □

Theorem 5. `Turan` $\bowtie$ `OI`

Proof. To see that `Turan` is not as strong as `OI`, consider the example used in the proof of Theorem 4. The domains being intervals, we know that `OI` computes the exact lower bound, 4. However the Turán heuristics gives us 2.

To see that `OI` is not as strong as Turán, consider the domains in Figure 6. The induced intersection graph $G_{\bar{X}}$ has 4 vertices ($n = 4$) and no edges ($m = 0$), thus `Turan` returns 4. However, the interval graph $G_{\bar{I}}$ induced by the same domains is a clique and then `OI` returns 1. □

Theorem 6. `MD` $\bowtie$ `OI`

Proof. To see that `MD` $\not\succeq$ `OI`, consider the interval graph in Figure (7,a) induced by the intervals of Figure (7,b). The exact independence number is 4 (for instance $\{v_2, v_3, v_8, v_9\}$ is an independent set of cardinality 4), and thus `OI` returns

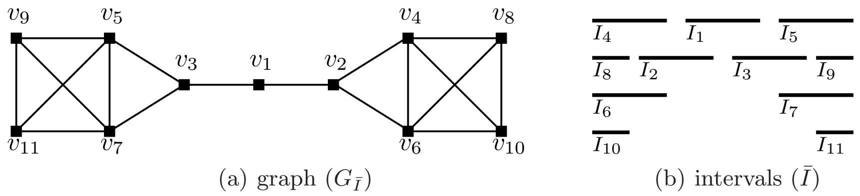


Fig. 7. Example for MD $\not\succeq$ OI.

4. However, the vertex with minimal degree is v_1, and no independent set of cardinality 4 involves v_1, therefore MD is not as strong as OI.

Figure 6 shows that OI $\not\succeq$ MD. The domains in Figure (6,a) induce a complete interval graph. So, OI returns 1. But the intersection graph is unconnected, so MD returns 4. Therefore OI is not as strong as MD. □

Theorem 7. LP $\succ$ MD*,* LP $\succ$ OI *and* LP $\succ$ Turan*.*

Proof. We first show that the value returned by LP is greater or equal to $\alpha(G_{\bar{X}})$. Consider a maximum independent set A of the intersection graph. We know that any two variables in A have no value in common. However for each variable $X_i \in A$ we have: $\sum_{v \in D(X_i)} y_v \geq 1$. Since the domains of those variables are disjoint, we have:

$$\sum_{v \in \bigcup_{X_i \in A} D(X_i)} y_v \geq |A| = \alpha(G_{\bar{X}})$$

And thus the total sum to minimize is greater than or equal to $\alpha(G_{\bar{X}})$. However, recall that OI, MD and Turan all approximate $\alpha(G_{\bar{X}})$ by giving a lower bound. Therefore LP is as strong as OI, MD and Turan. Moreover, the variables $X_1 \in \{1,2\}, X_2 \in \{2,3\}, X_3 \in \{1,3\}$ constitute an example showing that LP is strictly stronger, as the optimal sum for LP is 1.5, whilst $\alpha(G_{\bar{X}}) = 1$. □

Figure 8 summarizes the relations between the consistencies covered in this section.

7 A propagation algorithm for the NValue constraint

A template for an approximate propagation algorithm for NValue is given in Algorithm 3. In this template, one may use any of the methods described in the previous sections to replace the procedure `approx` (line 2). Notice that in Algorithm 3 (and in Algorithms 4, 5 and 6, we assume that the procedure stops and a backtrack occurs in the event of a domain being wiped out. The pruning on N is straightforward (lines 3 and 4). When we have $max(N) < min(N)$, there is clearly an inconsistency, $D(N)$ is wiped out, and the algorithm fails. In the following subsections, we consider the cases of lines 5, 6 and 7, where some filtering may be achieved. All other cases (line 8) satisfy the preconditions

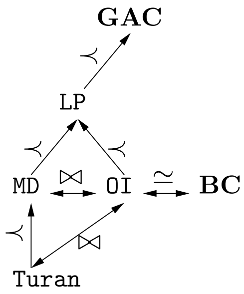


Fig. 8. Relations between pruning performed by the different algorithms on the AtMostNValue constraint.

of Lemma 2. Therefore, either the constraint is GAC, or we are unable to deduce any inconsistency because the lower bound lb for $card{\downarrow}(\bar{X})$ is not tight enough. In the rest of this section we use lb (resp. ub) to denote the value of $card{\downarrow}(\bar{X})$ (resp. $card{\uparrow}(\bar{X})$) used in algorithm 3. However, whilst ub is exact, lb is an approximation.

Algorithm 3: `Nval-pruning`

Data : $\bar{X}$, N

$ub \leftarrow card{\uparrow}(\bar{X})$;
$lb \leftarrow$`approx`$(card{\downarrow}(\bar{X}))$;
$max(N) \leftarrow min(max(N), ub)$;
$min(N) \leftarrow max(min(N), lb)$;
case $(ub = min(N) = max(N) \neq lb)$: pruning from below;
case $(lb = min(N) = max(N) \neq ub)$: pruning from above;
case $(|D(N)| = 2$ *and* $min(N) + 1 < max(N))$: pruning from within;
otherwise return;

Fig. 9. Algorithm for propagating the NValue constraint.

7.1 Pruning from below

This pruning is triggered when $ub = min(N)$ and $lb < min(N)$. In this situation, we know that some assignments may have too small cardinality, and therefore some values may not participate in assignments of cardinality $ub = min(N)$, which is the only cardinality satisfying the constraint. Making AtLeastNValue GAC is then sufficient to make the whole constraint GAC as this corresponds to the first of the three possible cases discussed in the proof of Theorem 1. In this situation, we can use a polynomial procedure for enforcing GAC on the SoftAllDiff constraint [14].

7.2 Pruning from above

This is the dual case, we know that some assignments may have too large cardinality, and therefore some values may participate only in assignments of cardinality above $max(N)$. This corresponds to the second case of the proof of Theorem 1. Making AtMostNValue GAC is then sufficient to make the whole constraint GAC. If lb is a sharp approximation of $card\downarrow(\bar{X})$, then we achieve GAC. However, if $lb < card\downarrow(\bar{X})$, then $card\downarrow(\bar{X}) > max(N)$ and there is no solution, but we do not detect that fact.

In [1] two observations are made in order to prune $\bar{X}$ which are relevant when using `MD` to compute $min(N)$. First, let A be a set of variables that form an independent set of the intersection graph, and let $X_i \in (\bar{X} \setminus A)$ be assigned to a value v which does not belong to any domain in A. It follows that the minimum number of values required will be at least $\alpha(G_{\bar{X}}) + 1$. Hence we can prune the value v from the domain of X_i when N is equal to $\alpha(G_{\bar{X}})$. This way of pruning the variables can be used with `MD` as well as with `OI`. There are no further difficulties when going from interval graphs to intersection graphs. Consequently, given an independent set A, we can propagate the following constraint.

$$\forall X_i \in \bar{X},\ \exists X_j \in A\ s.t.\ X_i = X_j$$

Now, suppose that A' is another distinct independent set. Thus, we have:

$$\forall X_i \in \bar{X},\ \exists X_{j_1} \in A,\ \exists X_{j_2} \in A'\ s.t.\ (X_i = X_{j_1} \wedge X_i = X_{j_2})$$

Therefore, one can prune values in $\bar{X}$ by finding a set of independent sets $\mathcal{A} = \{A_1, \ldots A_k\}$. The set of consistent values $\mathcal{V}$ is defined as follows:

$$\forall A \in \mathcal{A}, U_A = \bigcup_{X_i \in A} D(X_i), \quad \mathcal{V} = \bigcap_{A \in \mathcal{A}} U_A$$

It may be difficult to compute *all* independent sets of cardinality equal to N. One must therefore find a set which is as large as possible. In [1], from the first one found with `OI`, each independent set that differs by only one vertex is deduced. This can be computed in linear time, without increasing the algorithm's complexity. As a result of the way $\bar{X}$ is pruned, the algorithm described in [1] does not enforce BC on AtMostNValue. The following domains are a counter example:

$$X_1 \in \{1,2\},\ X_2 \in \{2,3\},\ X_3 \in \{3,4\},\ X_4 \in \{4,5\},\ N \in \{2\}$$

Only the values 2 for X_1, X_2 and 4 for X_3, X_4 are bound consistent. However, the independent sets considered will be $\{X_1, X_3\}$ and $\{X_1, X_4\}$. Therefore, the values that are consistent are $\{1, 2, 4\}$. On the other hand, this way of pruning can make holes in domains. Therefore the level of consistency achieved on AtMostNValue is incomparable with bound consistency. Although they are not equivalent, one can easily derive a procedure to enforce BC from `OI`. To check the (say lower) bound of a variable X_i, we assign this bound to X_i and recompute

N. If the value of lb, after this assignment, is greater than N, this bound is not BC.

With algorithms that do not compute independent sets in order to get a lower bound on N, like the linear relaxation method or the Turán heuristic, we are in a different situation. One possibility is simply not to do any pruning and thus backtrack only when the domain of N is emptied. Alternatively, using any method to compute a lower bound, we can use a brute-force method for pruning. We implemented our filtering algorithms based on `Turan` and `LP` using the second approach. Consider a value v in $\bigcup_{X \in \bar{X}} D(X)$, then suppose that we commit to this value being used and compute the contingent lower bound on N. A value strictly greater than N for this new lower bound means that v is inconsistent and can be removed from any domain it appears in. This procedure can be iteratively applied until all values have been checked.

7.3 Pruning from within

This pruning is triggered when $|D(N)| = 2$, $lb = min(N)$, $ub = max(N)$ and $lb + 1 < ub$. This is the last of the three cases in the proof of Theorem 1. In this case AtMostNValue and AtLeastNValue can be GAC whilst NValue is not. However, when these conditions are met, the following constraints perform this extra filtering:

$$(\text{AtMostNValue}(Min, \bar{X}) \vee \text{AtLeastNValue}(Max, \bar{X}))$$

Min and Max are two extra variables. We have the following theorem:

Theorem 8. *If $(D(N) = \{card{\downarrow}(\bar{X}), card{\uparrow}(\bar{X})\}$ and $card{\downarrow}(\bar{X}) + 1 < card{\uparrow}(\bar{X}))$ then* NValue$(N, \bar{X})$ *is GAC iff both the decomposition and the above constraints are GAC.*

Proof. $(\Rightarrow)$ The case where $D(N) = \{card{\downarrow}(\bar{X}), card{\uparrow}(\bar{X})\}$ or $card{\downarrow}(\bar{X}) + 1 < card{\uparrow}(\bar{X})$ does not hold is covered by Theorem 1. Now suppose this condition holds, and there is a value $v_i \in D(X_i)$ which is not GAC for NValue. By definition, this implies that any assignment such that the i^{th} element is v_i has a cardinality different from $card{\downarrow}(\bar{X})$ and from $card{\uparrow}(\bar{X})$, since these values are in $D(N)$. Moreover, there is no assignment with cardinality above $card{\uparrow}(\bar{X})$ or below $card{\downarrow}(\bar{X})$. Therefore we deduce that any assignment $\bar{v}$ involving v_i is such that $card{\downarrow}(\bar{X}) < card(\bar{v}) < card{\uparrow}(\bar{X})$. Hence, if $Min = lb \wedge Max = ub$ holds, then v_i would be inconsistent for both AtMostNValue$(Min, \bar{X})$ and AtLeastNValue$(Max, \bar{X})$).

$(\Leftarrow)$ If a value v_i belongs to a support, i.e., an assignment whose cardinality is either lb or ub, then either AtMostNValue$(Min, \bar{X})$ or AtLeastNValue$(Max, \bar{X})$) or both are GAC. □

Hence, we simply assign N to lb, then we compute B_1, the set of values inconsistent for AtMostNValue. Similarly, we assign N to ub and compute B_2, the set of values inconsistent for AtLeastNValue, In both cases, we use the methods described in section 7.1 and 7.2. Once this is done, we restore the domain of N, and prune all values in $B_1 \cap B_2$. Notice that B_1 may be underestimated, hence we do not achieve GAC.

8 Implementation

In this section we detail the implementation of the filtering algorithms used in the experimental section (section 9). We give the pseudo code of `Turan` and `MD`.

8.1 Ordered intervals

We implemented `OI` following exactly the pseudo-code in page 8 of Beldiceanu, Carlsson and Thiel's technical report [2]. The intervals were ordered at each call using a quick sort algorithm. There is little need to consider incremental algorithm for sorting as all intervals may have been modified between two calls to the propagator.

8.2 Intersection graph methods

In this case we again observed that incrementally updating the intersection graph was an overhead more than an optimisation. We used instead a simple method to reduce the size of the graphs considered. Clearly any value actually assigned to a variable will be part of the total set of values. We therefore compute g_{var}, the set of grounded variables, as well as the corresponding set g_{val} of values assigned to variables in g_{var}. We have three interesting cases:

1. $|g_{val}| > max(N)$
2. $|g_{val}| = max(N)$
3. $|g_{val}| = (max(N) - 1)$

In the first case, there is obviously an inconsistency which results in a failure of the propagation algorithm. In the second case, the situation is simple, the maximum number of values has already been used, and thus any value outside g_{val} is inconsistent. The third case is slightly more tricky. We know that the variables in $\bar{X} \setminus g_{var}$ can only use up to one extra value. We define d_{var} to be the set of variables that have no intersection with g_{val}, i.e., $d_{var} = \{X \in \bar{X} \mid D(X) \cap g_{val} = \emptyset\}$. The domain of any variable X in $\bar{X}$ can be then reduced to $D(X) \cap (g_{val} \cup \bigcap_{Y \in d_{var}} D(Y))$.

If none of the previous cases hold, the lower bound on N is computed using either `Turan` or `MD`. However, notice that for a variable X whose domain intersects with g_{val}, assigning X to a value contained in g_{val} will not increase the distinct number of values used and is therefore always the "best" choice. Hence these variables do not need to be considered in the intersection graph. The effective size of the intersection graph can thus be greatly reduced. Algorithm 4 performs this preliminary step, i.e., computes g_{var}, g_{val} and d_{var} and checks the three cases above, before giving the hand to either `MD` or `Turan`.

In Algorithm 5, we use the Turán inequality to compute a lower bound on $card{\downarrow}(d_{var})$ as well as for pruning values of variables in $\bar{X}$. The lower bound on $card{\downarrow}(d_{var})$ is obtained by adding $|g_{val}|$ to the quantity $\frac{n^2}{2m+n}$ where n stands for the number of nodes and m the number of edges in the subgraph restricted

Algorithm 4: `AtMostNval-pruning`

Data : $\bar{X}$, N
$g_{val} \leftarrow \emptyset$;
$g_{var} \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
$V \leftarrow \bar{X}$;
foreach $X \in \bar{X}$ **do**
  if $|D(X)| = 1$ **then**
    $g_{val} \leftarrow g_{val} \cup D(X)$;
    $g_{var} \leftarrow g_{var} \cup \{X\}$;
$min(N) \leftarrow max(min(N), |g_{val}|)$;
$K \leftarrow max(N) - |g_{val}|$;
$d_{var} \leftarrow \bar{X} \setminus g_{var}$;
foreach $X \in \bar{X} \setminus g_{var}$ **do**
  if $D(X) \cap g_{val} \neq \emptyset$ **then** $d_{var} \leftarrow d_{var} \setminus \{X\}$;
if $K = 0$ **then**
  foreach $X \in \bar{X} \setminus g_{var}$ **do** $D(X) \leftarrow D(X) \cap g_{val}$;
if $K = 1$ **then**
  foreach $X \in \bar{X} \setminus g_{var}$ **do**
    $D(X) \leftarrow D(X) \cap (g_{val} \cup \bigcap_{Y \in d_{var}} D(Y))$;

Fig. 10. Shared procedure for propagating the AtMostNValue constraint.

Algorithm 5: `Turan-pruning`

Data : $\bar{X}$, N
`AtMostNval-pruning`$(\bar{X}, N)$;
$n \leftarrow |d_{var}|$;
$m \leftarrow 0$;
foreach $X \in d_{var}$ **do**
  $degree[X] \leftarrow |\{Y \mid X \neq Y \in d_{var} \ \wedge\ D(X) \cap D(Y) \neq \emptyset\}|$;
  $m \leftarrow m + degree[X]$;
$min(N) \leftarrow max(min(N), |g_{val}| + \frac{n^2}{m+n})$;
if $min(N) \geq max(N) - 1$ **then**
  foreach $v \in \bigcup_{X \in d_{var}} D(X)$ **do**
    $d_v \leftarrow \{X \mid X \in d_{var} \ \wedge\ v \in D(X)\}$;
    $n' \leftarrow n - |d_v|$;
    $m' \leftarrow m - \sum_{X \in d_v} degree[X]$;
    if $|g_{val}| + 1 + \frac{n'^2}{m'+n'} > max(N)$ **then**
      foreach $X \in \bar{X}$ **do** $D(X) \leftarrow D(X) \setminus \{v\}$;

Fig. 11. Algorithm based on the Turán inequality for propagating the AtMost-NValue constraint.

to d_{var}. Notice that in algorithm 5, m stands for twice the number of edges ($2m$ in the formula). Then, for every value in $\bigcup_{X \in d_{var}} D(X)$, the same computation can be made on the subgraph whose nodes correspond to variables that do not contain v. This value can be added to $|g_{val}|+1$ to get a lower bound on $card{\downarrow}(\bar{X})$ in the case where the value v was to be used. The value v is therefore pruned if the inequality $|g_{val}| + 1 + \frac{n^2}{2m+n} > max(N)$ holds.

Algorithm 6: `MD-pruning`

```
Data    : X̄, N
AtMostNval-pruning(X̄, N);
A ← ∅;
while |d_var| > 0 do
    n ← |d_var|;
    min = +∞;
    foreach X ∈ d_var do
        degree[X] ← |{Y | X ≠ Y ∈ d_var ∧ D(X) ∩ D(Y) ≠ ∅}|;
        if min > degree[X] then
            min ← degree[X];
            Y ← X;
    A ← A ∪ {Y};
    d_var ← d_var \ {Y};
    foreach X ∈ d_var do
        if D(X) ∩ D(Y) ≠ ∅ then  d_var ← d_var \ {X};
min(N) ← max(min(N), |A| + |g_val|);
foreach X ∈ X̄ do  D(X) ← D(X) ∩ (g_val ∪ ⋃_{X∈A} D(X));
```

Fig. 12. Algorithm based on the minimum degree heuristic for propagating the AtMostNValue constraint.

In Algorithm 6, we use the minimum degree heuristic to compute a lower bound on $card{\downarrow}(d_{var})$. The variables in $\bar{X}$ are pruned as described in section 7.2. We use the independent set constructed in the greedy elimination scheme for this purpose. If we find an independent set B, we concatenate it with g_{var}, that is, we compute the set $A = (g_{var} \cup B)$ and we propagate the following constraint:

$$\forall X_i \in \bar{X},\ \exists X_j \in A\ s.t.\ X_i = X_j$$

which in consequence reduces the domains of variables in $\bar{X}$ to their intersection with the set $g_{val} \cup \bigcup_{X \in A} D(X)$.

8.3 Linear relaxation

The linear programming approach was implemented with CPLEX. In this case, the linear program was incrementally modified during search as it is too costly to create a new linear program each time we call the propagator. The linear program is simply updated to reflect the pruning that occurred since the last call. If a modification occurred, the linear program is solved to obtain a lower on $card\!\downarrow\!(\bar{X})$. When this lower bound is greater or equal than $max(N) - 1$, we use the following filtering procedure: For each variable $y_v = 0$ (representing a value) in the optimal solution, we solve `LP` $\cup\{y_v = 1\}$, then we remove the value v from all domains if and only if the solution to this linear program has an objective value greater than $max(N)$.

9 Experimental results

In this section, we compare empirically the different approaches introduced in this paper with Beldiceanu's bounds consistency algorithm. The purpose of this experimental setting is twofold. We first aim at assessing the significance of the theoretical comparison developed in this paper, then we want to evaluate the tradeoff between filtering power and asymptotic cost. We used two types of benchmarks. First, in order to isolate the testing of these propagators from other modelling issues, we used randomly generated instances that we augmented with an AtMostNValue constraint. Second we compared the algorithms on a structured problem, namely, finding the dominating set of the *Queen's graph.*

9.1 Random Problem

We randomly generated instances of binary CSPs then we added an AtMost-NValue constraint and solved the resulting problems using the four methods discussed in this paper (`OI`, `Turan`, `MD` and `LP`). In solving such problems, the AtMostNValue constraint is exposed to a wide range of different variable domains. The problem instances are generated according to model B [15], and can be described with four parameters: the number of variables n, the domain size d, the number of constraint m and the number of forbidden tuples t per constraint. We add a fifth parameter k corresponding to the maximum number of values allowed by the AtMostNValue constraint. All instances are solved using the classical *domain/degree* variable ordering heuristic, and lexicographical value ordering. We generated 500 instances of the following 5 classes of random binary CSP:

$$\begin{aligned}
class\ A &: \langle n = 100, d = 10, m = 250, t = 52, k = 8\rangle \\
class\ B &: \langle n = 50, d = 15, m = 120, t = 116, k = 6\rangle \\
class\ C &: \langle n = 40, d = 20, m = 80, t = 240, k = 6\rangle \\
class\ D &: \langle n = 200, d = 15, m = 600, t = 85, k = 8\rangle \\
class\ E &: \langle n = 60, d = 30, m = 150, t = 350, k = 6\rangle
\end{aligned}$$

Classes A,B and C are at the phase transition, i.e., nearly half of the instances have a solution with the required number of values. Classes D and E are underconstrained. We report the mean cputime and number of backtracks to solve these instances or prove unfeasibility in table 1 for classes A,B,C and table 2 for classes D,E. We also report the number of calls to the filtering method for the AtMostNValue constraint and the total cputime spent specifically on filtering this constraint.

	OI	Turan	MD	LP
	class A $\langle n = 100, d = 10, m = 250, t = 52, k = 8\rangle$			
cputime (total)	3,358 ms	3,262 ms	**2,858 ms**	9,342 ms
#backtracks	20,530	21,650	16,955	**16,735**
#calls	26,112	27,332	22,102	**21,827**
time (filtering)	430 ms	173 ms	**162 ms**	4,074 ms
time (per call)	0.06561 μs	**0.02530** μs	0.02926 μs	0.74379 μs
	class B $\langle n = 50, d = 15, m = 120, t = 116, k = 6\rangle$			
cputime (total)	6,950 ms	7,211 ms	**2,596 ms**	15,812 ms
#backtracks	72,773	90,786	18,885	**17,866**
#calls	130,227	161,444	36,333	**34,491**
time (filtering)	1,421 ms	762 ms	**287 ms**	10,038 ms
time (per call)	0.04349 μs	**0.01881** μs	0.03149 μs	1.15960 μs
	class C $\langle n = 40, d = 20, m = 80, t = 240, k = 6\rangle$			
cputime (total)	5,769 ms	5,872 ms	**1,672 ms**	11,998 ms
#backtracks	62,992	77,072	12,564	**11,467**
#calls	115,720	140,901	25,017	**22,994**
time (filtering)	1,142 ms	564 ms	**183 ms**	7,839 ms
time (per call)	0.03935 μs	**0.01595** μs	0.02918 μs	1.35824 μs

Table 1. Instances at the phase transition.

	OI	Turan	MD	LP
	class D $\langle n = 200, d = 15, m = 600, t = 85, k = 8\rangle$			
cputime (total)	12,721 ms	21,494 ms	**44 ms**	307 ms
#backtracks	77,848	410,514	233	**212**
#calls	140,531	745,457	626	**590**
time (filtering)	8,742 ms	7,022 ms	**11 ms**	227 ms
time (per call)	0.12417 μs	**0.01880** μs	0.03736 μs	0.76833 μs
	class E $\langle n = 60, d = 30, m = 150, t = 350, k = 6\rangle$			
cputime (total)	124 ms	141 ms	**4 ms**	56 ms
#backtracks	1,469	3,800	33	**29**
#calls	2,999	7,477	142	**134**
time (filtering)	60 ms	33 ms	**1 ms**	45 ms
time (per call)	0.04059 μs	**0.00888** μs	0.01735 μs	0.67299 μs

Table 2. Under-constrained instances.

For both problems at the phase transition, and under-constrained problems, we observe the fastest solution times by propagating the AtMostNValue constraint using the `MD` method. Whilst this achieves less pruning than the `LP` method, the difference in nodes explored is typically slight. The additional overhead of the `LP` method does not justify the small amount of additional pruning it achieves. On the other hand, the `OI` and `Turan` methods achieve much less pruning for similar computational cost.

9.2 Structured Problem

We compared the same propagation algorithms on the problem of finding a dominating set of the *Queen's graph* with minimum cardinality. The Queen's graph of order n, denoted Q_n, has one vertex for each square of a $n \times n$ chessboard. There is an edge between two vertices if and only if a queen on one vertex can attack the other vertex. A dominating set hence corresponds to a set of squares such that if a queen is put on each of them, all squares either contain a queen or can be attacked. The dominating set of Q_6 is given as example in figure 13. This problem can be modeled with a single instance of an AtMostNValue

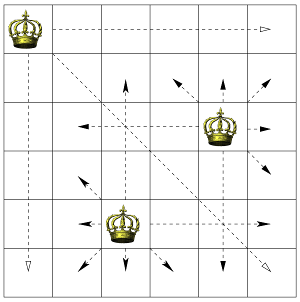


Fig. 13. A dominating set of Q_6.

constraint. Given the graph Q_n, we introduce n^2 variables $X_1, \dots X_{n^2}$ and n^2 values $v_1, \dots v_{n^2}$, both standing for squares of the chessboard. The value v_j belongs to the domain of X_i if and only if the edge (i, j) belongs to Q_n. Finally we post an AtMostNValue constraint on $X_1, \dots X_{n^2}$. Clearly, if a solution using at most N values exists, then the queens on the corresponding squares (values) can attack all other squares (variables). In [13], it is shown that for $n \leq 120$, all minimum dominating sets for the Queen's graph have cardinality either $\lceil n/2 \rceil$ or $\lceil n/2 + 1 \rceil$. We therefore solved instances only for these two values of N. We report the cputime, number of backtracks, number of calls to the filtering method and the total cputime spent specifically on filtering this constraint for solving these instances or prove unfeasibility in table 3.

Instance	Statistics	OI	Turan	MD	LP
$Q_6 : 3$	cputime (total)	0.02 s	0.02 s	**0 s**	0.01 s
(SAT)	#backtracks	520	370	**0**	**0**
	#calls	1,162	922	53	**37**
	time (filtering)	0.01 s	0.01 s	**0 s**	0.01 s
$Q_7 : 4$	cputime (total)	8.6 s	2.2 s	**0.01 s**	0.36 s
(SAT)	#backtracks	104,333	121,051	270	**28**
	#calls	209,954	239,558	676	**117**
	time (filtering)	7.2 s	1.4 s	**0.01 s**	0.36 s
$Q_8 : 5$	cputime (total)	73 s	13 s	**0 s**	0.25 s
(SAT)	#backtracks	623,029	637,031	101	**0**
	#calls	1,263,692	1,261,241	340	**69**
	time (filtering)	61 s	7.9 s	**0 s**	0.25 s
$Q_8 : 4$	cputime (total)	$>$ 300 s	$>$ 300 s	36 s	**11 s**
(UNSAT)	#backtracks	-	-	880,669	**2,243**
	#calls	-	-	1,785,601	**4,873**
	time (filtering)	-	-	30 s	**10.8 s**
$Q_9 : 5$	cputime (total)	$>$ 300 s	$>$ 300 s	203 s	**16 s**
(SAT)	#backtracks	-	-	4,076,033	**3,628**
	#calls	-	-	8,328,122	**8,078**
	time (filtering)	-	-	172 s	**15.69 s**

Table 3. Dominating Set of the Queen's Graph.

We observe that the algorithms introduced in this paper, in particular the propagation method using the minimum degree heuristic (`MD`) and the linear relaxation (`LP`) perform dramatically better than the previous method, that is, the bounds consistency algorithm (`OI`). We can also observe that the extra filtering achieved by the linear relaxation outweighs its higher computational cost on the largest instances.

10 Related work

The maximum independent set is a well known problem in graph theory and a number of approximation algorithms have been proposed. We used two simple and intuitive algorithms for the sake of simplicity and because `MD` is successful in practice. However, algorithms with better approximation ratio exist, for instance see [11]. Any such algorithm may replace `MD` into the propagation algorithm.

We have seen that the linear programming approach is always stronger, even than a complete method for finding a maximum independent set. It is difficult to identify where the linear relaxation for the minimum hitting was first introduced, as it is such a simple model. It is certainly given in [8]. One weakness of the linear programming approach is that it is difficult to deduce which values to prune when $min(N) = max(N)$. Indeed, other methods, whilst computing a lower bound, also compute an independent set that can be used for pruning $\bar{X}$, whereas the LP does not provide such an independent set.

11 Conclusion

Propagating generalized arc consistency on the NVALUE constraint is NP-hard. In order to filter inconsistent values, one has to obtain tight bounds on the number of distinct values used in assignments. Whilst the upper bound can be obtained in polynomial time with a maximal matching procedure, the lower bound alone is NP-hard to compute. Therefore, our focus is on methods which achieve lesser levels of consistency. A procedure proposed by Beldiceanu considers domains as intervals, which allows the independence number of the induced interval graph to be computed in polynomial time. The independence number of this graph is a valid lower bound on the number of distinct values. We introduce three new methods for approximating this lower bound. The first two approximate the independence number of the intersection graph. However, these algorithms have a quadratic worst case time complexity, and do not guarantee a tighter lower bound. The last approach is to use a linear relaxation of the minimum hitting set problem. The cardinality of the minimum hitting set is a tight lower bound on the number of distinct values. This always finds a tighter lower bound than the approaches based on the maximum independent set problem. Experiments show that the approximation method based on a greedy approximation to the maximum independent set offers a good tradeoff between pruning and runtime.

References

1. N. Beldiceanu. Pruning for the *Minimum* Constraint Family and for the *Number of Distinct Values* Constraint Family. In Toby Walsh, editor, *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP-01)*, volume 2239 of *Lecture Notes in Computer Science*, pages 211–224, Paphos, Cyprus, 2001. Springer-Verlag.
2. N. Beldiceanu, M. Carlsson, and S. Thiel. Cost-Filtering Algorithms for the two Sides of the Sum of Weights of Distinct Values Constraint. Technical report, SICS report T2002-14, 2002.
3. C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. Tractability of Global Constraints. In Mark Wallace, editor, *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP-04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 716–720, Toronto, Canada, 2004. Springer-Verlag. Short paper.
4. R. Debruyne and C. Bessière. Some Practicable filtering Techniques for the Constraint Satisfaction Problem. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 412–417, Nagoya, Japan, 1997. Morgan Kaufmann.
5. M. Dincbas, P. van Hentenryck, H. Simonis, and A. Aggoun. The Constraint Logic Programming Language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 693–702, Tokyo, Japan, 1988.

6. P. Roy F. Pachet. Automatic Generation of Music Programs. In Joxan Jaffar, editor, *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming (CP-99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 331–345, Alexandria, VA, USA, 1999. Springer-Verlag.
7. O. Favaron, M. Mahéo, and J.-F. Saclé. Some Eigenvalue Properties in Graphs (Conjectures of Graffiti - ii). *Discrete Mathematics*, 111:197–220, 1993.
8. S. Shahar G. Even, D. Rawitz. Hitting Sets when the VC-dimension is Small. *to appear in Information Processing Letters*, 2004.
9. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, 1979.
10. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 439–448, Montréal, Québec, Canada, 1994. ACM Press.
11. V. Th. Paschos M. Demange. Improved Approximations for Maximum Independent Set via Approximation Chains. *Applied Mathematical Letters*, 10:105–110, 1997.
12. E. Marzewski. Sur deux Propriétés des Classes d'Ensembles. Fund. Math., 33:303–307, 1945.
13. P. R. J. Östergard and W. D. Weakley. Values of Domination Numbers of the Queen's Graph. *Electronic Journal of Combinatorics*, 8:1–19, 2001.
14. T. Petit, J.C. Régin, and C. Bessiere. Specific Filtering Algorithms for Over-Constrained Problems. In Toby Walsh, editor, *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP-01)*, volume 2239 of *Lecture Notes in Computer Science*, pages 451–463, Paphos, Cyprus, 2001. Springer-Verlag.
15. P. Prosser. Binary Constraint Satisfaction Problems: some are harder than others. In Anthony G. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 95–99, Amsterdam, The Netherlands, 1994. John Wiley and Sons, Chichester.
16. J.C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, Seattle, WA, USA, 1994. AAAI Press.
17. P. Turán. On an Extremal Problem in Graph Theory. *(in Hungarian),* Mat. Fiz. Lapok, 48:436–452, 1941.